IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

DECLARATION UNDER 37 C.F.R. §1.131

Applicant:	Botz, *et al.*	Docket No.:	ROC920000271US1
Serial No.:	09/818,064	Group Art Unit:	2154
Filed:	03/27/01	Examiner:	VU, VIET DUY

TITLE: APPARATUS AND METHOD FOR MANAGING MULTIPLE USER IDENTITIES ON A NETWORKED COMPUTER SYSTEM

Mail Stop AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

In response to the office action dated 06/25/05, the named inventors make this declaration for the purpose of swearing behind the date of the Cole patent application publication US2002/0093857 cited by the examiner in the pending office action.

The effective date of Cole is the filing date of the provisional patent application upon which Cole depends, namely 12/07/2000. As inventors of the claimed invention, we declare that the invention was actually reduced to practice before 12/07/2000.

Claim 1 recites:

1. An apparatus comprising:

at least one processor;

a memory coupled to the at least one processor;

first software residing in the memory and executed by the at least one processor, the first software including a first user registry that contains a first user identity for a selected user that is used to authenticate the selected user to the first software;

second software residing in the memory and executed by the at least one processor, the second software including a second user registry that contains a second user identity for the selected user that is used to authenticate the selected user to the second software; and

an identity mapping mechanism that provides a mapping between the first user identity and the second user identity.

The claimed invention was actually reduced to practice on or before September 12, 2000, before the effective date of the Cole patent application publication. Exhibit A shows a document entitled "Server Group Enterprise User Identity Mapping Design Proposal" which bears a date of 09/12/2000. This document shows actual reduction to practice for the claimed invention.

Because this declaration establishes invention of the subject matter of the rejected claims prior to the effective date of the Cole reference, we respectfully request that the rejection based on the Cole reference cited by the Examiner be withdrawn.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

(1) Full name of the first inventor: Patrick S. Botz

Signature: Patrick S. Botz Date July 19, 2005

(2) Full name of the second inventor: Patrick Jerome Fleming

Signature: Patrick Jerome Fleming Date July 18, 2005

(3) Full name of the third inventor: Timothy James Hahn

Signature: Date July 20, 2005

Server Group Enterprise User Identity Mapping

Design Proposal

Author:
Pat Botz, Rochester botz@us.ibm.com
Contributors
AIX:
Lee Terrell, Austin, terrell@austin.ibm.com
OS/400:
Pat Fleming, Rochester, fleming@us.ibm.com
Kyle Henderson, Rochester, kyleh@us.ibm.com
John McMeeking, Rochester,jmcmeek@us.ibm.com
OS/390:
John Dayka, Poughkeepsie, dayka@kgnvmc.vnet.ibm.com
Rich Guski, Poughkeepsie, guski@us.ibm.com
Tim Hahn, Endicott, hahnt@us.ibm.com
SWG:
Marlena Erdos, Raleigh, marlena@us.ibm.com

Approved By:

Name	Organization	Date
	Austin	
	Poughkeepsie	
	Rochester	

Contents

Purpose

Server Group has been participating in the Enterprise User/Identity workgroup headed up by Lee Rafalow. We are in general agreement with that proposal. The purpose of this paper is to serve as a high-level implementation document for the OS/390, AIX, and AS/400 Server Group platforms.

Background

IBM customers with multiple platforms would like an easier way to manage users in an enterprise. Today, users must be defined separately on every platform in the enterprise. Customers spend a lot of time and money managing users throughout the enterprise. They want a cheaper solution.

With Windows 2000 (Win2K), Microsoft is providing a single point of user administration capability for their operating systems. The task is much less complex for Microsoft as they only need to support two operating systems, NT and Win2K.

The challenge for IBM Server Group is to provide similar functionality across the disparate Server Group platforms. The IBM operating systems are vastly different since they were designed at different times, by different people, with different requirements. This makes the problem much more complex and limits the possible solution space.

The first thing IBM needs to do is stop apologizing for having different operating systems. The Server Group operating systems help solve real problems and issues facing IBM customers in today's IT environments. For example, security is the second biggest concern of CIOs today and is projected to become the top concern among CIO's after January 1, 2000. Security is one of the primary value propositions of all Server Group platforms. IBM and Server Group should leverage these strengths -- sure we're different than Microsoft, and with good reasons.

Rather than blindly trying to implement a Microsoft-like solution, Server Group proposes attacking the problem by providing the necessary infrastructure and APIs for all IBM platforms on which Software Group can build an "Enterprise User Management" product. The same infrastructure will also allow customers and business partners to write mulit-tier applications which never have to ask the user for a platform-specific user identity and password when accessing legacy data. Furthermore, the data will continue to be protected by IBM's highly respected platform-specific security services. Finally, the infrastructure will eliminate the need to cache or send passwords over the network. The infrastructure necessary to accomplish all this is described in this paper.

The Enterprise User Management product will be a single-point-of-control, graphical user interface (GUI), through which an administrator can manage all of a user's "identities" on all systems in the enterprise. Further, this product will hide the different operating system semantics from the administrator.

The framework and APIs necessary to build the Enterprise User Management product are the same as those that business partners and customers will use to build multi-tier, cross-platform applications. The infrastructure on which the APIs are based relies on two technologies: Kerberos

and LDAP. Kerberos provides single-signon, single-password, authentication advantages; LDAP provides a single point of management for users' identities, along with a common repository for managing the relationship between each user and each of his or her local identities.

There is a natural affinity between Enterprise User Identity Mapping and the technology provided by Dascom/Policy Director. This proposal focuses on user identification and authentication while the Policy Director focuses on authorization. See Appendix C for more discussion on this topic.

Operating System Requirements

Operating system requirements are fundamentally different than application requirements. The primary reason for this is that operating systems must ensure "operating system integrity." Simply put, operating system integrity means that the operating system is able to enforce operating system defined security semantics for the manipulation of all operating system resources. If the operating system cannot guarantee this for all operating system resources, then it cannot guarantee the security of any application or application data running on that operating system.

The need for operating system integrity drives most of the requirements listed in this section. The following are the requirements:

- **Provide platform interoperability with Win2K in an Intranet environment**
  Win2K is using Kerberos as it's distributed authentication mechanism in the Intranet environment. The server group platforms need to provide inter operability with Kerberos to maintain seamless client-server applications. Current customer frustrations with the need for either caching of passwords or multiple passwords can be alleviated by operating system and application exploitation of Kerberos for authentication.
- **Enable multi-tier applications across IBM platforms**
  Most major IBM accounts have more than one IBM platform. Each platform has useful enterprise information. Sharing that information seamless at the application level is cumbersome because each platform has it's own architected concept of a user. It is not feasible to re-architect all platforms to share a single user registry. However, by adding an identity mapping abstraction, we can provide the same capabilities as a shared-single-user registry.
- **Provide a set of APIs common to each platform that allow applications to map from an authenticated identity for an individual or entity to another identity for that same individual or entity**
  Kerberos principles have different characteristics than OS defined users. The differences include the length of the name, the format of the name, and the inclusion of a Kerberos defined "realm" name. The OS platforms need to provide a set of APIs which allow applications on those platforms to easily determine the local user represented by a Kerberos principle. In addition, to provide ease of management, APIs will also be necessary to find mappings from one arbitrary user identity to another, from an ePerson to a particular identity or all identities associated with that ePerson, from an arbitrary identity to it's associated ePerson.
- **Maintain the integrity of each OS**
  OS integrity means that OS defined security semantics must be adhered to manage operating system resources. Users are a fundamental OS resource. If the OS defined security semantics can be circumvented by users or applications, then ALL customer data on that OS is vulnerable no matter how good the security within the applications run on that OS.
- **Changes made via legacy mechanisms to a local user must be reflected from an enterprise view and vice-versa**
  Customers have millions of lines of code and dollars invested in JCL, REXX, CL, and SCRIPT files that use legacy user management interfaces. A common user management solution cannot require that this investment be abandoned. Alternatively, changes made from the enterprise view must also be reliably made on the appropriate platforms/environments.

Synchronization issues (e.g. the enterprise view has a different data than the local view) must be avoided when those issues can cause operating system integrity exposures. We must be able to ensure that the data in the enterprise view always matches the data in a local view.

- **Create customer value by providing an infrastructure on all server group platforms on which a cross-platform common user management interface can be built**
  Since we cannot eliminate the need for each OS to define and manage user's for that OS, we must provide a relatively easy mechanism to manage an individual's multiple identities in the multi-paltform environment. One of the objectives of this cross server group platform infrastructure would be to, over time, homogenize user identity across server group platforms in a manner similar in concept to what the DASCOM IntraVerse ACL engine and APIs do for the authorization space.
- **When managing a user (create, change, delete, identity mapping) in a particular system/environment, maintain the security semantics of the system/environment in which that local user is defined**
  Security semantics for managing a user includes items such as privileges which are necessary to be able to manage users for that OS. Each OS defines it own semantics. Applications that don't provide the same semantics create OS integrity exposures. This is acceptable when a customer creates the application and the exposure -- i.e. customer choice. It is not acceptable for operating systems or IBM applications to create these exposures because these exposures are forced by IBM on the customer and their enterprise.
- **Hide the differences in security semantics of each platform behind a cross-platform/environment common user management tool**
  While the underlying security semantics of each OS must be maintained when managing a local user on that system, the mechanism we create for central management must hide the semantic differences. Essentially this means that an administrator using a central user management tool must ultimately have an identity with the appropriate priviliges/authority on each system on which identities are being managed. While not ideal, this only needs to be setup once and is necessary to maintain OS integrity.
- **Avoid the need to re-write/re-architect legacy OS systems**
  User definition is an integral part of OS architecture. Providing a common user definition and semantics across all IBM platforms would require re-architecting nearly all of each of the OS's. Not only is this not feasible for IBM from a cost and time point of view, it would also destroy our customer's investment in any IBM platforms.
- **Leverage the strengths and value propositions of each platform**
  Each of the platforms have slightly different value propositions. If major pieces of the value proposition for a platform are compromised, there will be no need for a common solution which includes that platform.
- **Hide the mechanics of the directory from legacy systems**
  While the directory is an obvious choice for providing a solution, the solution should not require a large set of new semantics for administrators to deal with.
- **Treat the relationship between an individual -- or entity -- and all of his/her -- or its' -- identities as an operating system resource**
  Each operating system which can manage the relationship between a person and all of his or her identities must be able to enforce it's semantics for manipulating these relationships. The relationship between a person and his or her identities are an operating system resource.

- **Performance**
  The performance of the identity mapping APIs will be dependent on the implementation of the LDAP infrastructure for managing identity relationships, which includes network interaction. We recognize that we can not quantify network latency, or the effect of the load in the performance of the LPAP server's response time at this point in time. Therefore, while no performance targets have yet been set, we recognize that performance of the entire authentication step including identity mapping must be as fast as possible, as this "effective path length" may be in the critical path of a user-to-application request.

Terms

It is useful to define several terms which are used throughout this paper.

"*platform*," refers to computer hardware and the corresponding operating system (i.e. OS/390, AS/400, AIX, Win2K, etc).

"*Environment*" refers to application products that choose to implement an application specific concept of a user (e.g. Lotus, WebSphere, etc).

Note that digital certificates sort of straddle the line between being an operating system resource and an application resource. For the purposes of this paper, assume they are included in either or both definitions.

"*Local user*" or "*local user identity*" is a platform or environment specific user as defined in that platform or environment's user registry.

"*User*" generally refers to the human being associated with a user identity.

"*Enterprise user*" or "*eUser*" refers to the LDAP schema and class definition defined by the Secureway Directory team. An enterprise user represents a single individual or entity within the enterprise -- it does not represent an operating system resource and is stored in an LDAP. The information associated with an enterprise user is assumed to be telephone white-pages-like information.

"*Local group*" or "*local group identity*" is a platform or environment specific mechanism used to refer to sets (or groups) of local user identities. Typically these are defined in the platform or environment user (or user and group) registry.

"*Enterprise group*" or "*eGroup*" refers to the LDAP schema and class definition for groups defined at the enterprise level.

"*Local identity*" refers to either a "local user identity" or a "local group identity" depending on the context of the discussion.

Enterprise User Infrastructure Description

This proposal introduces a new operating system resource called "identity relationships." Identity relationship data is the information about the relationship between all of a user's local identities and the user's enterprise user entry. The process of determining a specific local user identity given a different specific local user identity is called "*identity mapping*."

Figure 1 shows a logical view of the identity relationship information. It depicts logically how, given one identity for a specific user, any other related identity can be found. While not specifically shown, a single user may have multiple identities on a single platform or environment. The infrastructure supports this. Digital certificate identities are also assumed to be supported by the infrastructure.

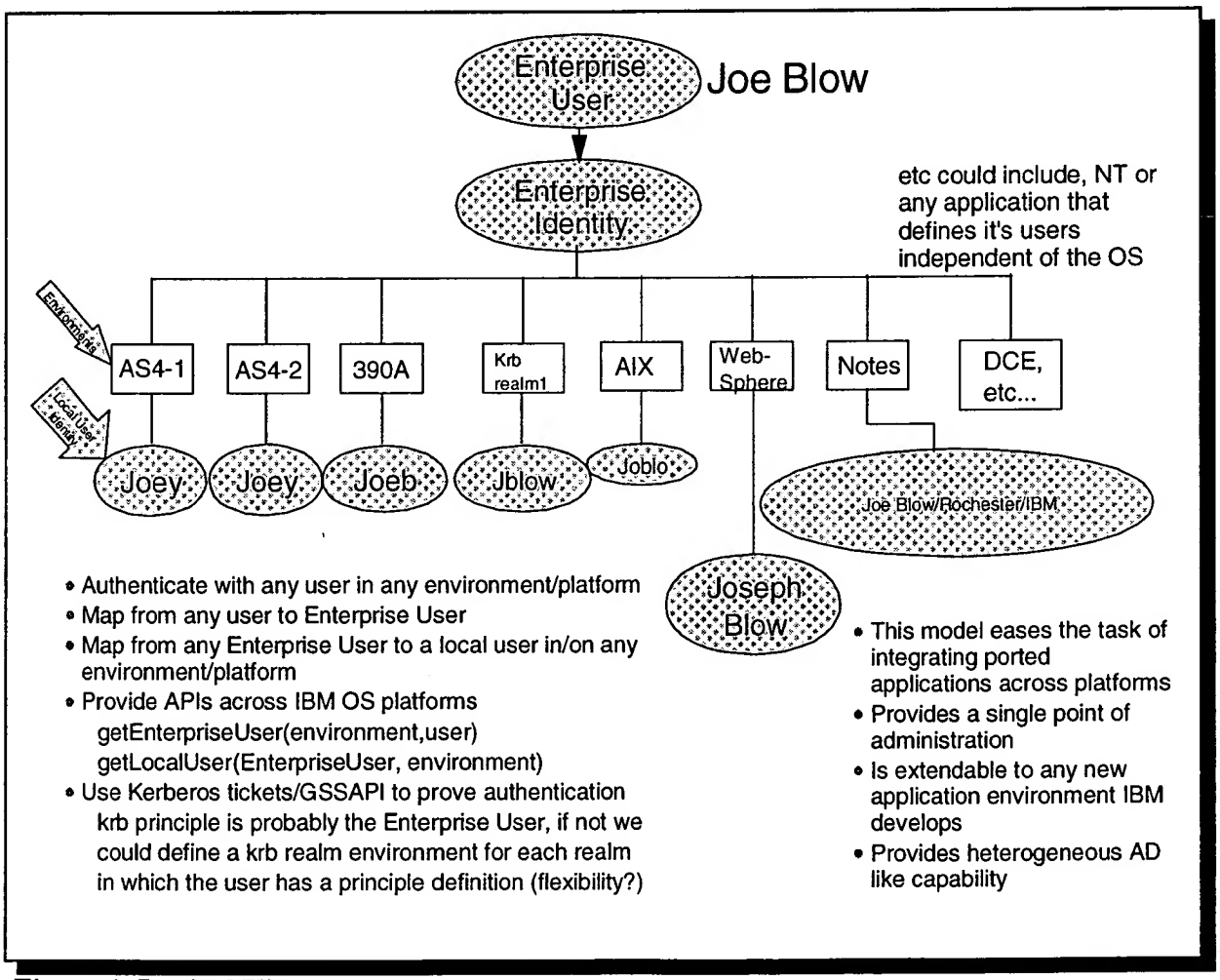

Figure 1. Logical View

In practice, most identity mapping will be from a Kerberos principle identity to the local identity of platform that received the Kerberos service ticket. For example, refer to figure 1, below. Assume a user, *Joe Blow*, logs into his Windows 2000 desktop. He automatically gets a Kerberos credential for his realm -- realm1 -- which identifies Joe Blow with a principle name of *jblow*. When *Joe Blow* requests services from the AIX system which is also named AIX, a Kerberos

service ticket which identifies the requester as *jblow@realm1* is passed as authentication. The application providing the service on system AIX, receives the service ticket and using the infrastructure defined in this paper determines that the local user identity on this system related to principle *jblow@realm1* is the AIX user name, *joblo*. If the AIX application chooses, it can do a setuid() operation before accessing any resources to ensure that user requesting the service is authorized to the necessary resources. Even though there are multiple identities, they all refer to the same user.

There are uses other than mapping from Kerberos principle to local user identity. For example, mapping from a Lotus Notes ID to a local user identity. The infrastructure and the associated APIs allow for all mappings.

High-Level Architecture

Figure 2, is a graphical view of the infrastructure. The infrastructure is based on two technologies: Kerberos and LDAP. These technologies are used both individually and in combination to help meet all of the requirements described above. One of the primary design points for the infrastructure is to never store the information twice. Storing information only once avoids nasty synchronization issues and thus meets the requirement that changes made via either a legacy or enterprise user mechanism are automatically available via the other mechanism.

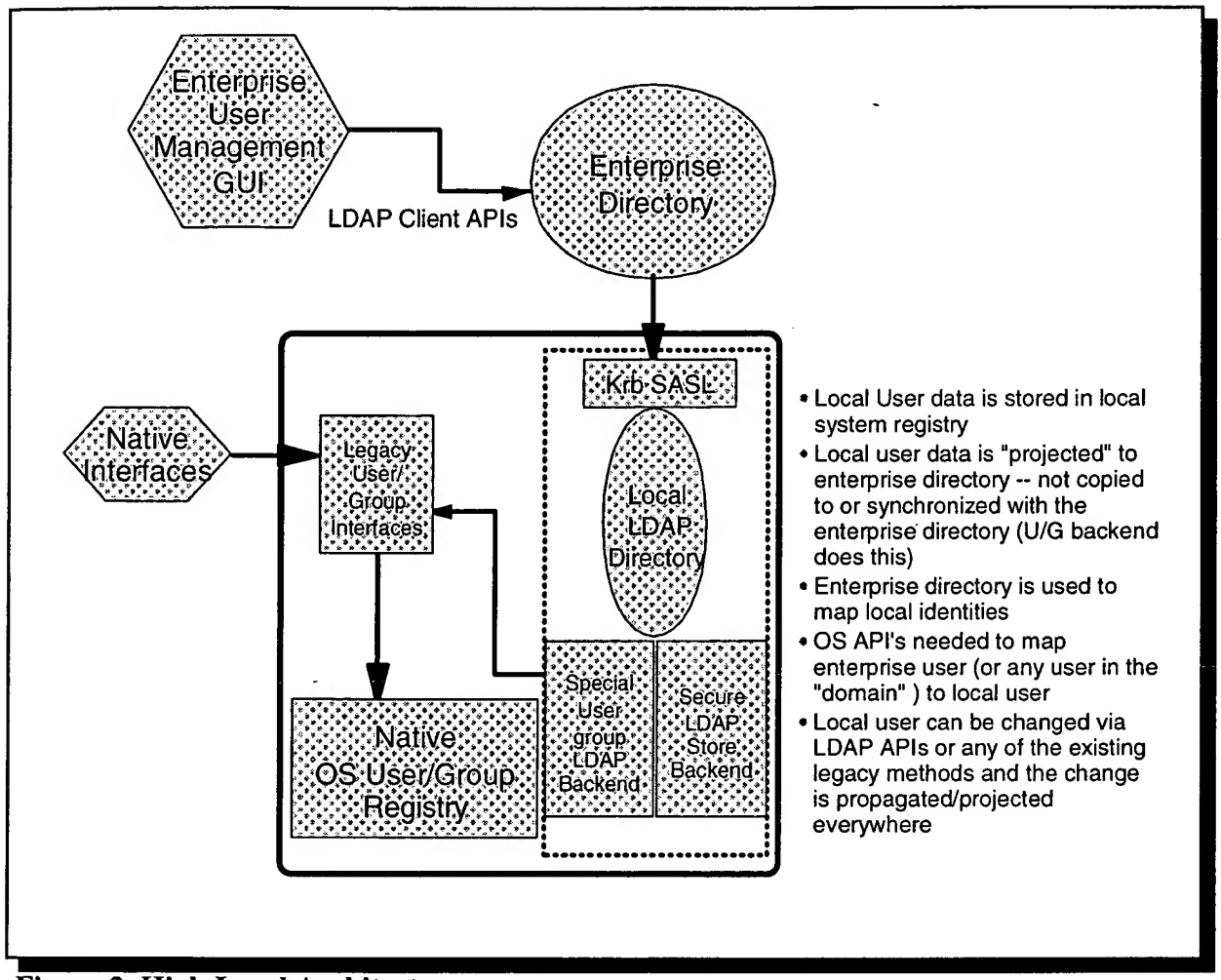

Figure 2. High Level Architecture

Architecture Usage of LDAP

LDAP technology is used in a couple of different ways. First, just the protocol is used to provide for manipulation of local identities and attributes in a distributed environment. LDAP is useful because it is really two technologies: the LDAP protocol, and the LDAP repository. The LDAP protocol is very useful as a single access mechanism. Providing a single storage mechanism via an LDAP repository is not as important, especially as it relates to managing operating system resources across heterogeneous platforms.

Because it would be difficult and expensive[1] to remove local user registries from platforms, the infrastructure "projects" the local identity into the LDAP namespace. Essentially, the LDAP client APIs are used to view and modify local user information, but the information is never stored in an LDAP repository. Instead, the local user data "appears" to be in the repository. This is accomplished by supplying a specialized user and group back end for the Secureway Directory on each server group platform. The special user and group back end is depicted in the dashed box in

[1] Expensive in terms of both implementation cost and performance pathlength.

figure 2. Note how the back end ties into the local user registry interfaces to access and update data. The LDAP protocol provides the distributed access without storing the data directly in an LDAP repository.

Next, a **secure LDAP repository** is used to store and manage the **identity relationships** for all enterprise users. To provide a single point of management we need to be able to manage the relationship between a user and all of his or her identities on/in all platforms/environments in the enterprise. This **identity relationship information is considered an operating system resource** and as such requires a secure LDAP repository. A secure repository is implemented via another specialized back end as depicted in the dashed box in figure 2, above. It is, in most respects, the same as the "normal" Secureway Directory repository but access to the data in this back end is always done under a local identity associated with the user requesting the access to the data. This is accomplished via a SASL bind mechanism that supports Kerberos. The secure back end should be implemented in an open and extensible manner because other operating systems uses, outside the scope of identity mapping are also envisioned.

Identity mapping relationship information can be managed centrally or replicated to other machines. The security semantics of the platform on which the information being requested is stored are always enforced. This means that if an AS/400 relies on the identity mapping information in a directory which resides on AIX, then the security semantics on AIX would determine whether the requested operation is allowed. In practice, we assume that the identity mapping information will be replicated to many platforms in the enterprise. A multi-master directory is not required for the infrastructure but would provide the most useful functionality. If the directory does not have multi-master capability, then additional information regarding the location of the single master will be necessary to implement the enterprise user management product.

Architecture Usage of Kerberos

Kerberos tickets are used as a distributed authentication mechanism. They, along with identity mapping, help provide single-signon functionality. The ability to map from a Kerberos principle to a local identity provides interoperability between IBM platforms and also with **Windows 2000**.

Each platform or environment is free to choose the registry it requires for user definition. The identity mapping infrastructure provides a mechanism for relating all identities for a user in the enterprise with each other and with the user's enterprise user definition.

Figure 3, (best viewed in color), shows the different security semantics enforced for manipulating different components of the identity mapping infrastructure. Note particularly how the security semantics of the registry which defines a local identity are enforced for accessing or manipulating that local identity. Also note that the security semantics enforced for accessing or manipulating the identity relationship information is a function of the platform on which the information is being accessed or manipulated.

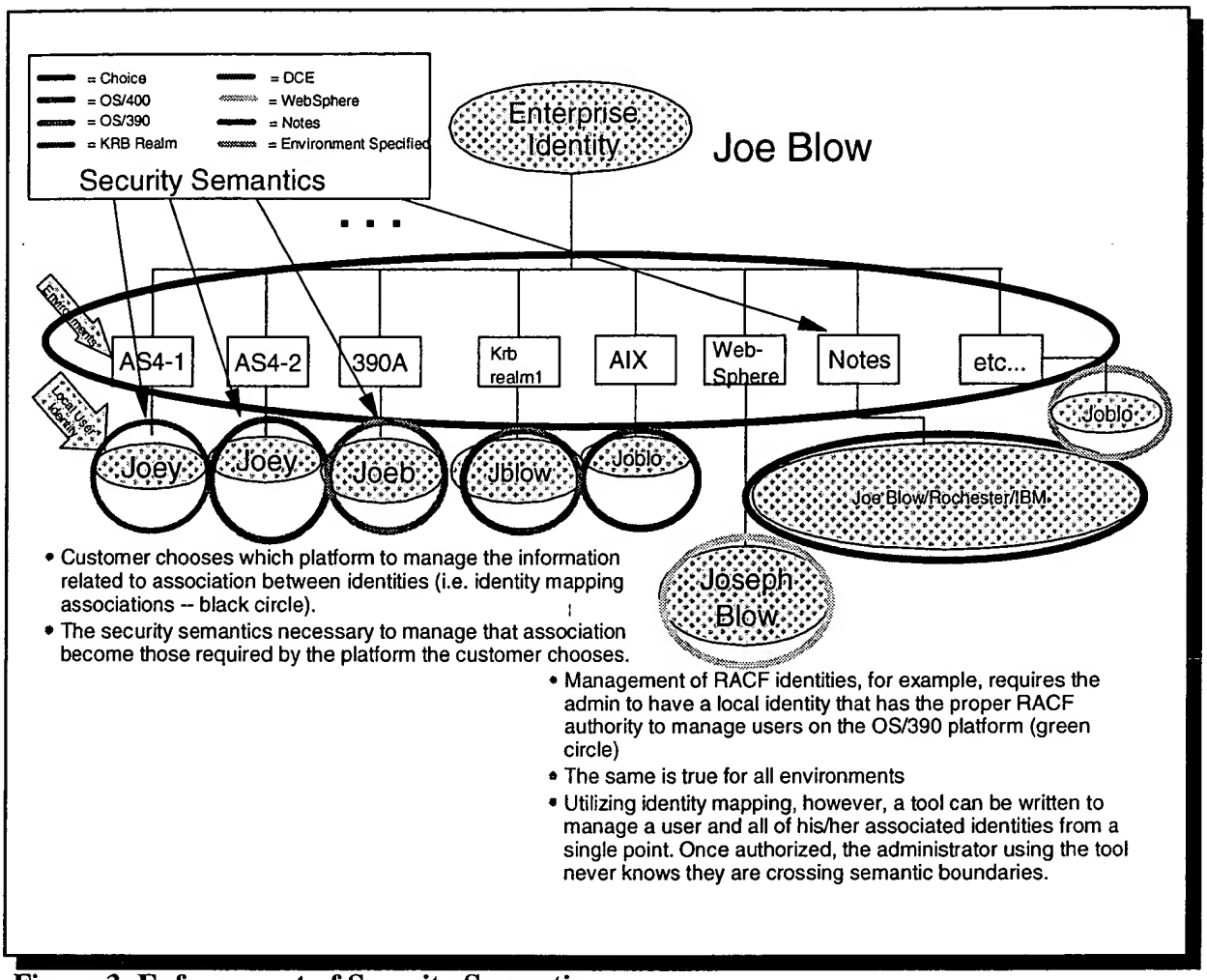

Figure 3. Enforcement of Security Semantics

See appendix A, for a lower level description of the LDAP schema for enterprise user identity mapping.

Enterprise Groups

Enterprise user identity mapping helps applications and administrators deal with N-tier authentication issues and management of users in an enterprise. All operating systems in common use today also provide the concept of groups of users. For any Enterprise User Management product to be successful it must also deal with management of groups and group membership -- both enterprise and local -- in the enterprise.

The purpose of an enterprise group, is to provide a single GUI interface for management of group membership across the enterprise. For example, the notion of membership in "board of directors" may be important to multiple environments in the enterprise. Naturally, the membership of "board of directors" should be consistent across all environments. How do we achieve this? While an "easy answer" is to create a single LDAP-accessible group, it is not a viable solution for the same reasons shared-single-user registry is not viable for user identities on Server Group platforms.

A better answer is to use the LDAP repository to manage both enterprise user membership in enterprise groups and the relationship between an enterprise group and associated local group identities. Then use the LDAP protocol to directly manage local user membership in local groups in each specific environment. By building on the identity mapping infrastructure described above, we can do just this.

While an enterprise user and its associated local user identities are a set of relationships for a specific user, an enterprise group, on the other hand, contains the relationship between zero or more *enterprise users* **and** *associated local group identities.* When an administrator manages the enterprise group -- adding or deleting members to or from it -- those management actions will be reflected in all the local group identities associated with the enterprise group.

Figure 4 shows an enterprise group "GroupA." GroupA has a membership list containing a list of enterprise users and a list of associated local group identities (AS1/ASGRP1). Using the same mechanisms as used for local user identities (specialized user and group back end, figure 1), local group identities are "projected" into the LDAP namespace. The local group identities can then be accessed via LDAP client APIs using the enterprise group name (GroupA).

Enterprise Group Scenario

An enterprise administrator wishes to add an existing enterprise user, MM, to the enterprise group "*platinum*". MM has local identities on at least two systems AS1 and AS3. These local user identities are "*as1mm*" and "*as2mm.*" There are local groups on each system named *as1grp* and *as2grp* respectively. Assume these groups are associated with enterprise group *platinum.*

When the administrator adds an enterprise user, MM, to enterprise group *platinum* the admin tool retrieves a list of MM's local identities The admin tool then retrieves a list of *platinum*'s local group identities. Remember that local identities include the system or environment name. The logical intersection of the system/environment names is taken by the admin tool. That is, it strikes from the list it got from the enterprise user lookup, any system/name pair where that system does not have a local group identity associated with *platinum.*

If MM had local identities on platforms AS1, AS2, and AS3, but only AS1 and AS3 were represented in the list of associated local group identities, the resulting list would consist of AS1 and AS3. Note that the list also contains the local user identity for MM on AS1 and AS3 also because that information is part of the identity mapping information for a user (e.g. AS1:*as1mm*, AS2:*as2mm*)

The admin tool would then do the following: Under the administrator's identity, it would add enterprise user MM to the list of enterprise users that are members of enterprise group *platinum*. Of course, the administrator identity must have appropriate authorization to do this. Then, using the administrator's Kerberos identity, the admin tool would contact each system identified in the step above and request via LDAP client APIs to add MM's local user identity on that platform to the local group identity associated with the enterprise group *platinum*. The operation is now complete.

Note that we do not require an enterprise group member to have identities on all the systems that have an associated local group identity. In fact, we allow the enterprise group member to not have an identity on any of the systems at all. (We expect that each member will have identities on most of the systems participating in the enterprise group. We just don't require it.)

Note that this approach helps us with synchronization issues. Having a membership list in the enterprise group lets us "fill up" a new local group that is participating in the enterprise group so that it is in synch with all the other participating local groups. It may also be beneficial to have an eUser's entry contain a list of their eGroups to allow us to appropriately update systems when that eUser gets a new local identity. Here's the (simplified) algorithm:

```
for each eGroup in the (e)person's list
    if the new local identity is on a system that is participating in the eGroup then
        add the new local identity to that system's local group.
```

The admin tool must allow the administrator to update a single system participating in the enterprise group with a user's local identity on that system. We need this function to allow an administrator to "manually" synchronized the set of systems participating in an enterprise group. The need for manual synchronization occurs when an attempt to add or delete an (e)person from an enterprise group fails to add/delete that person's local identity to a local group on one or more systems. This could happen due to network failures or a participating system being down.

The admin tool will notify an administrator "immediately" if an addition or deletion to a local system was not successful. This is necessary because membership in a group confers authority to the members of that group. Timely "correction" of a group might be vital for the safety of a system. If an update could not be performed, the administrator must know as soon as possible so that s/he can take remedial action.

At the time a local group is associated with an enterprise group, it will be possible to also update the membership of the local group. This process involves generating a list of local users that are members of the local group. Using this list, each local user's enterprise user identity can be found

using the identity mapping infrastructure. Each enterprise user can then be added to the enterprise group. The process of automatically removing membership in an enterprise group when a single local group identity is disassociated with the enterprise group is more complicated but can also be done.

In terms of figure 4, if an administrator added enterprise user "User1" to enterprise group "GroupA," the following would happen:

- Enterprise group "GroupA" would be updated to reflect "User1" as a member
- Local identity Joe on system AS1 would be add to local group ASGRP1 on system AS1
- Local identity Fred on system AIXZ would be added to local group AIXGPZ on system AIXZ

The local groups on systems AS1 and AIXZ are updated because "User1" has a local identity on these systems and these systems have local groups associated with enterprise user "GroupA."

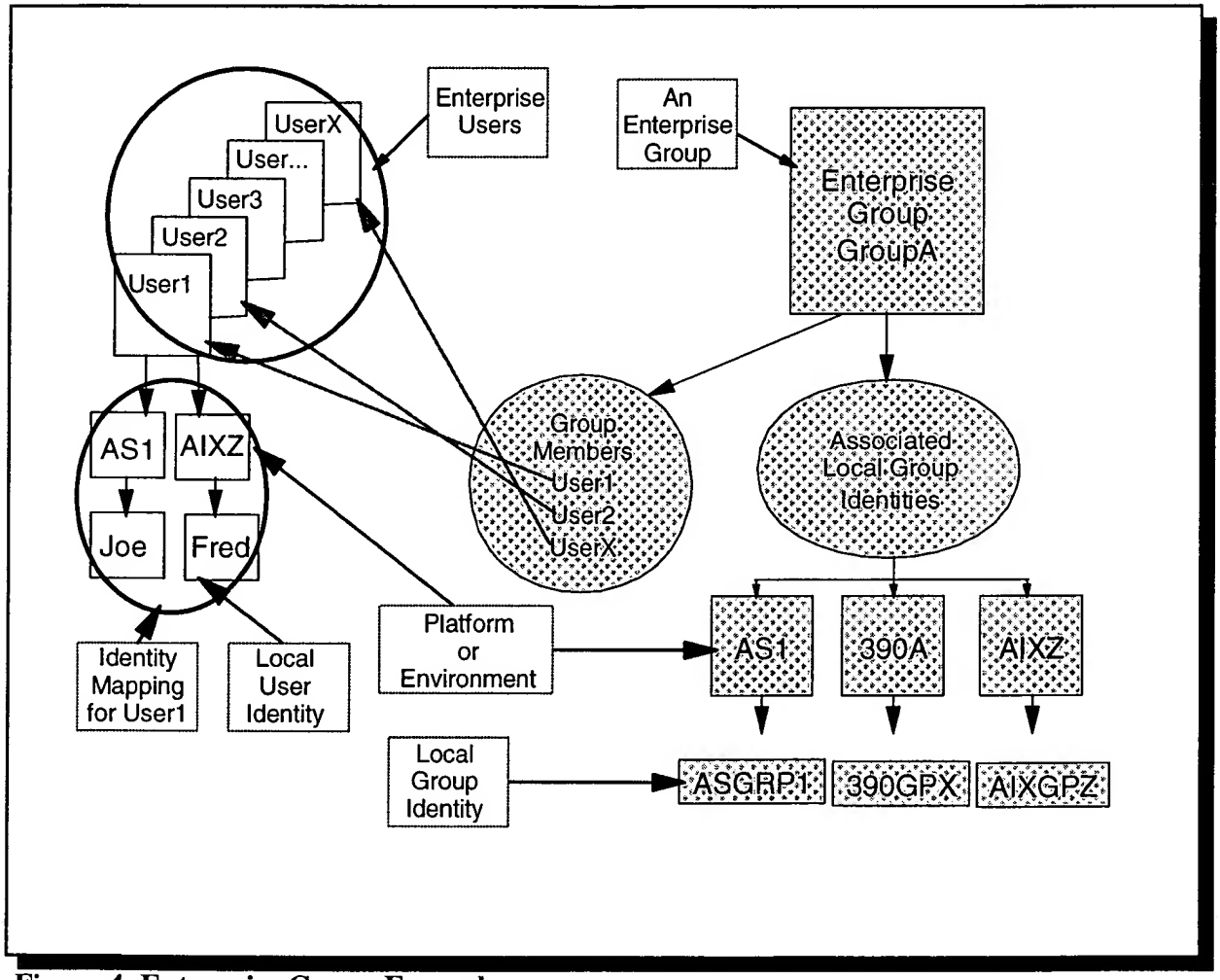

Figure 4. Enterprise Group Example

See Appendix B for a lower level description of the enterprise group LDAP schema.

Finally, note that this proposal does not address nested groups. This is primarily because nested groups are not supported on all platforms (e.g. AS/400, OS/390). We haven't explored the

implications of supporting nested groups at the enterprise group level while at the same time not supporting them on participating platforms.

Identity Mapping APIs

Operating systems function and applications utilizing identity mapping will not deal directly with the directory or client LDAP APIs. Instead, each platform will provide a set of APIs that hide the infrastructure. The code that implements the APIs will use the LDAP client APIs.

The following are the loosely defined APIs and descriptions of their parameters and behaviors they would perform. The first API is more formally defined than the rest. The descriptions of the rest of the APIs are less formally defined and rely on the description of the first.

The intent here is to define behavior not the C prototype. Note that while the parameters are defined below as strings, this is only done to facilitate the description of the behavior of the API.

```
int getAssociatedIdentity(char *known_identity,
                          char *target_identity_environment,
                          char *buffer,
                          int buffer_size,
                          error_indicator)
```

Example invocation and usage:

```
int count;
char buffer[1024];
count = getAssociatedIdentity( "KERBEROS:principle@realm", "OS400/rchas02", buffer, 1024,
errno)
if (count < 0)
  { error indicated by errno }
else
  if (count = 0) // none found
    { no associated identifier of the specified target environment }
  else
    if (count = 1)  // exactly one found...this may or may not be what was expected...depends on
the app
      { do the right thing }
     else
       { for (i=0; i< count; i++) {   // more than one found...may or may not be what was
expected...depends on app
           do the right thing by looping through the buffer contents
           (e.g. find the one you want or iterate through all of the returned vaules)
         }
       }
```

Given a known identity, this API returns all of the associated identities for the specified target environment

This API first retrieves the info from the system configuration object that tells the API 'where' the mapping server is. It then formulates an LDAP request to the mapping server to find known_identity. Once the known identity is found in the identity mapping tree (eMappedUser I think we're calling it now) it does whatever other LDAP protocol is needed to find the UIDs of all associated identities of the target_identity_environment.

known_identity = a string of the form *environment : identifier*. Where *environment* identifies the "*Type*" of identifier followed by a colon and the specific identity, *identifier*, within that environment. *Type* can be of the form *type,* or *specific_type_instance,*where *type* is a string representing the platform or environment (e.g. KERBEROS, X509CERT, etc). *specific_type_instance* is a string which is of the form *type/instance* representing the type and a more specific instance of that type (e.g. OS390/systemA...the last part could be a TCP/IP addr instead). The format of the *identifier* field will depend on the value of the environment field. For Kerberos it would be principle@realm. For OS400 it would be a name. For AIX it might be a string representation of the numeric UID value (e.g. AIX/cadsys4). Note: the actual format is open for debate. If there is a more efficient way to represent the info then lets do that. Example values for known_identity would be: OS400/rchas03:botz, KERBEROS/rchrealm:pbotz, X509CERT:cn=Pat Botz,ou=Rochester,o=IBM,st=MIN,c=US, AIX/rchland:botzy, OS390/sysplex5:mrbotz, etc. I know that the Kerberos form is not the standard principle@realm format, but we could easily parse it into this form. Alternatively, we could special-case Kerberos to allow for KERBEROS:pbotz@rchrealm or a URL-style perhaps.

target_identity_environment = a string of the form *specific_type_instance.* Follows the same rules as above. Example values = KERBEROS/realm7.

Note: there may be a variety of convience APIs that perform the action but assume certain types. For example, we might provide an API such as, getLocalIdentity(known_identity). This API would assume the target_identity_environment was the local system's OS type and TCP/IP name. Other than that it does the same thing as the API defined above.

Given the scenario where a user has multiple identities for a given target_identity_environment, it might be useful to have an API that takes an additional parameter that specifies data that should be used to further specify the type of identity requested. For example,

```
int getSpecificAssociatedIdentity(char *known_identity,
                                  char *target_identity_environment,
                                  char *additional_properties,
                                  char *buffer,
                                  buffer_size,
                                  errno)
Example invocation =  count = getSpecificAssociatedIdentity("KERBEROS:principle@realm",
                                                             "OS400/system1",
                                                             "administrator",
                                                              buf,
                                                              2048,
                                                              errno);
```

This would return only those identity mappings on system1 that had an additional property of administrator. We should leave the possible values for this field loosely defined. Platforms may suggest properties that they will recognize, but applications should be free to be able to define whatever they want for this field also.
I'm not sure what the properties might be for any platform/environment. I know I want these values to only have meaning within the eMappedUser as opposed to being copies of specific local identity properties such as *SECADM privilege for an AS/400 user profile (synchronization problems again). These vaules should be as free form as possible.
int **getAssociatedIdentityCount**(known_identity, target_identity_environment);
returns the number of target environment identities associated with the known identity

int **getIdentities**(known_identity, char *buffer, int buffer_size, errno)
returns all associated identities for a known identity

int **getIdentitiesCount**(known_identity)
returns the number of all associated identities for a known identity

char ***getEnterprisePerson**(known_identity).
Given a known identity, this API returns the associated ePerson name.
E.g. char name[512] = getEnterprisePerson("OS390/sysplex6:botzy");

The following set of APIs are useful if you start with the ePerson rather than an identity....

int **getIdentityByEnterprisePerson**(ePerson_name, target_identity_environment, char *buffer, int buffer_size, errno)
Given a known ePerson name, return the identity(s) for the given target environment.
E.g. count = getEnterprisePersonIdentity("Patrick S. Botz", "AIX/austinland", results, 512, errno);

int **getEnterprisePersonAssociatedIdentityCount**(ePerson_name, target_identity_environment)
Given a known ePerson name, return the number of associated identities for the target environment -- same number as returned by getIdentitiesCount...just derived differently

int **getEnterprisePersonIdentities**(ePerson_name, buffer, buffer_size, errno);
Given a known ePerson name, return all of the associated identities

int **getEnterprisePersonIdentitiesCount**(ePerson_name);
Given a known ePerson name, return number of all of the associated identities

These are the basic APIs we believe applications would need to be able to deal with identity mapping. Some may have been missed, but these should provide the general idea of the functionality that applications will need.

<need to define an additional set of group related APIs, get local group identities from enterprise group, get members of group, get enterprise group from local group, others?>

Performance

The authentication step from an application point of view includes receiving a Kerberos ticket (gss_accept_context, etc), retrieving the principle from the ticket, and calling an identity mapping API to retrieve the local identity represented by the Kerberos principle. The identity mapping API requires a bind with the identity mapping LDAP repository which authenticates the user requesting the information, an impersonation of the local identity of the requesting user, a search of the repository for the authenticating user's Kerberos principle, and finding the requested identity associated with that Kerberos principle.

It is assumed, that a replica of the identity mapping repository will be kept on all platforms where performance is high priority. For example, an MQ Series platform will probably perform many identity mapping requests and would therefore require a local replica of the identity mapping repository. A machine used mostly for interactive, development, or test workloads (as opposed to a production server) may not require a local replica. Whether or not a local replica is required should be a configuration option for the system administrator.

Customer Value

The customer value in this proposal comes from the applications and products that can built on this infrastructure -- not from the infrastructure itself. Server group looks to SWG to provide these products. Without the infrastructure, however, it will be nearly impossible for SWG to provide a tool that meets all of the requirements described above.

The primary SWG product is a common user management GUI that is used by system administrators to manage users throughout the enterprise. This tool would allow a single "user administrator" to manage the creation, changes, and removal of user identities through a single GUI interface.

Very interesting applications such as **data mining** can also be built on top of the infrastructure. Consider a large enterprise that has OS390, AS/400, AIX, and Windows platforms. Data on each of these systems is secured according to whatever security scheme was used when the data was created. When initially created it was probably not known that information on SystemA would be useful when combined and analyzed with data on SystemB. Each system has existed for a relatively long period of time and each has a separately managed set of local users and the each set of data has an existing security scheme. By utilizing the identity mapping infrastructure and API's, applications can be written to perform data mining analysis **without having to change how the data is secured on each system**. If enterprise user Fred has a local identity on SystemA that has authority to the appropriate information on SystemA and Fred also has a local identity on SystemB that has authority to the data on SystemB, then Fred can use a multi-tier application that accesses both sets of information without changing the security scheme of the data on either system.

The **value** of not having to analyze, change, or implement new security mechanisms **can not be minimized**. By utilizing only the native security on each platform the customer avoids having to manage both native and application level security schemes. Adding application level authority or policy on top of native authority also causes problems with synchronization and auditing.

Using the Infrastructure to Provide Customer Value

As stated earlier, the infrastructure itself does not directly provide customer value -- the products that exploit the infrastructure do. The infrastructure is necessary to provide a common set of APIs across IBM Server Group platforms. Further, the infrastructure allows the commonality without sacrificing the value proposition of any Server Group platforms.

SWG, business partners, and customers will rely on the APIs to write multi-tier, cross-platform applications. These applications can access legacy data and allow the customer to rely on existing security policy for protecting that data.

Enterprise User Management Product

The enterprise user management product would rely on LDAP client API's and the Kerberos authentication mechanism to provide a single common user interface for all users in an enterprise across all IBM platforms.

The tool would allow an administrator to deal mostly with enterprise users. For a given user It would have, at least, the following functions:

- Display all local identities for this user
- Add local identity for a specific system. The attributes to be supplied for the local identity would vary depending on the type of system
- Change an attribute for a local identity
- Remove a local identity for an enterprise user
- Associate an existing local identity with a user
- Disassociate a local identity from an enterprise user

While not necessary, it may be useful to treat Kerberos identities separately in the interface. A Kerberos identity is semantically different from a local identity in that all platforms participating in a Kerberos realm can recognize a Kerberos principle, while only a particular platform will recognize a local identity.

There is a genesis problem that will have to be solved. Before an administrator can manage users on a specific system, the administrator must have a local identity on that system and the system must be able to map a Kerberos principle to that local identity. One way to handle this is to provide platform specific interfaces that allow the creation of relationships between a local identity, a Kerberos principle, and an enterprise user. Native user management interfaces would be enhanced to allow the administrator to specify the associated Kerberos principle and enterprise user. The native interfaces would use either LDAP client APIs or API wrappers to add these relationships.

To allow an administrator to use the common user management product, the following steps must be accomplished:

- Configure the system to be participate in a Kerberos realm
- Configure the system to use an enterprise identity "server"
- A local administrator uses native interfaces to add the relationships described above to the identity mapping server
- The common user management tool can now be used by this administrator to manage user's on this system.

Theoretically, at least, IBM could provide the identity mapping function on Windows 2000. Similarly, IBM could provide the identity mapping function on any UNIX(tm) platform. If these assumptions are correct, then, theoretically, the common user management tool could support all platforms in the enterprise whether IBM supplied or not.

Figures 5 and 6 below are intended as an example of what an enterprise user manager GUI might look like.

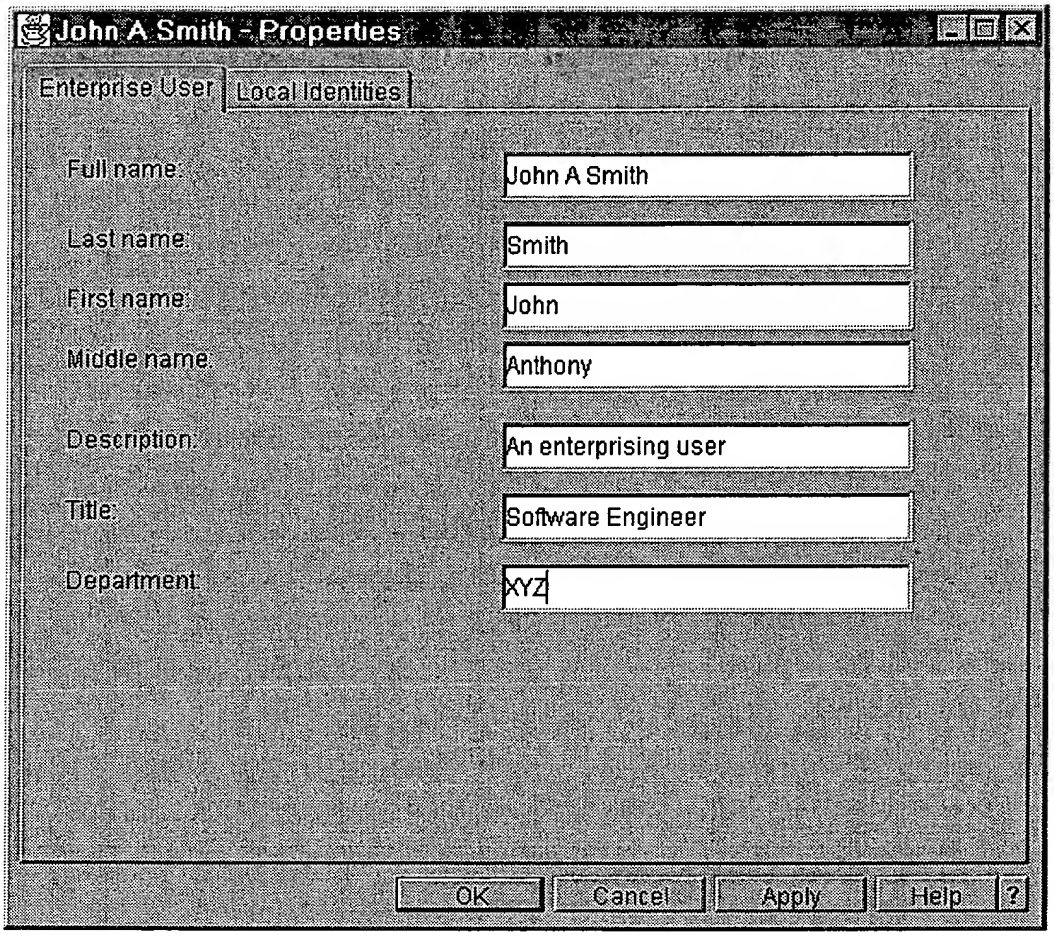

Figure 5. Example Enterprise User Manager

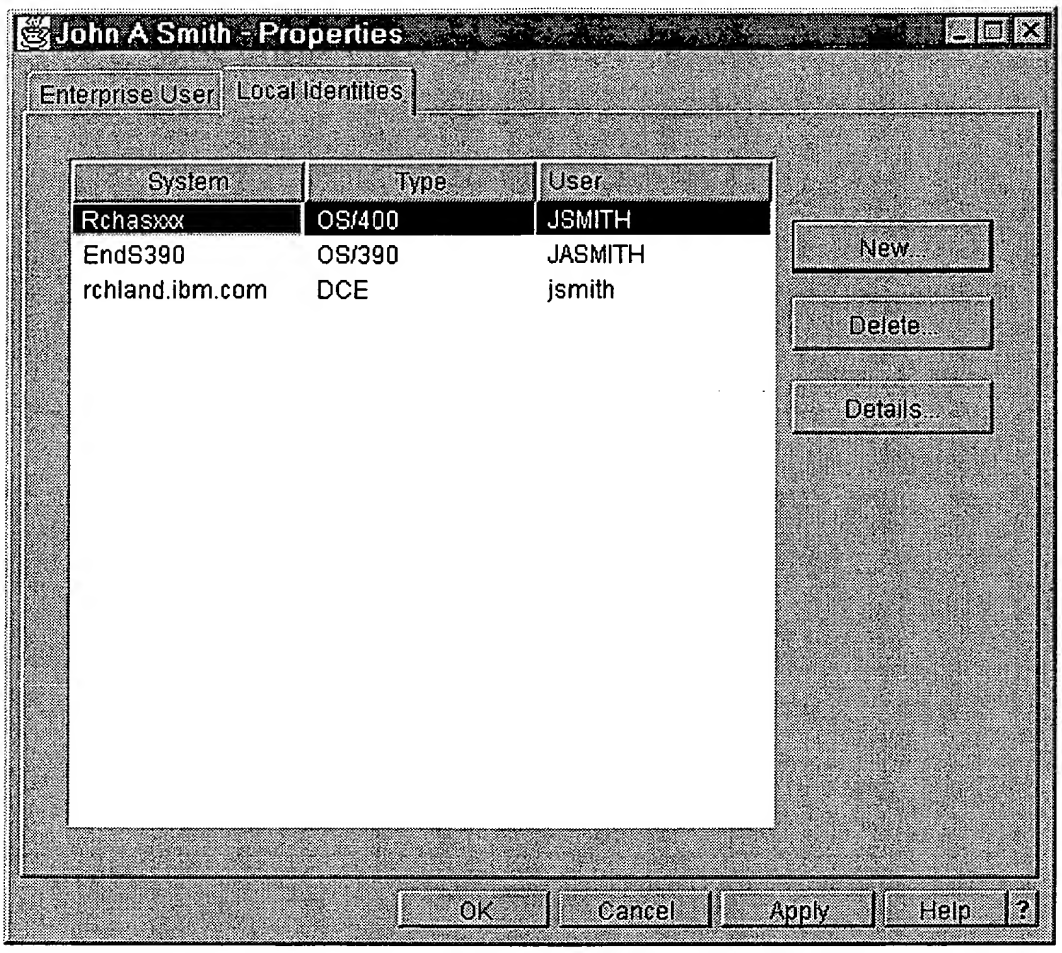

Figure 6. Example Enterprise User Manager -- continued

Implementing Multi-tier, Cross-Platform Applications

Given the APIs proposed above, it will be relatively easy to create multi-tier, cross-platform applications for IBM platforms and Windows 2000. The initial-tier application presumably provides the user interface. When it needs information or an operation performed on another IBM platform, it acquires a delegatable Kerberos ticket for the appropriate service on that platform.

Using application defined protocols and the GSS API's, the Kerberos ticket is passed to the middle-tier. The middle-tier application uses the identity mapping APIs to find and impersonate the user on the middle-tier platform. This allows the application to securely and appropriately access information on the middle-tier without changing how the information is secured.

If the middle-tier application requires information from yet another platform, it can use the GSS APIs to request the service from the corresponding third-tier application on behalf of the original user. This is possible since the Kerberos ticket it received was delegatable. The third-tier application would perform the same steps as the middle-tier did to impersonate the user on the local platform before accessing resources secured on the local platform.

Appendix A

LDAP Identity Mapping DIT & Schema

Introduction

This section covers the LDAP schema related to the identity mapping proposal and introduces several new object classes and attribute types, directory management API's and recommendations for the structure of a directory namespace (DIT). Covered are; several schema related terms, diagrams of example directories, suggestions of some possible directory management functions and finally details of new object class and attributes.

The identity mapping schema proposal attempts to satisfy the following requirements:

1. To utilize the existing IBM schema definitions where possible. This is accomplished by reusing the ou, person, eUser, eAccount object classes and several attribute types.
2. To provide organization within the directory for managing user information. This is accomplished by providing a default DIT structure (use of directory entries as containers).
3. To allow LDAP directory servers to "project" environment information. That is, to allow access to local resources via the LDAP protocol as if those resources were stored in an LDAP repository. This requires the DIT to be partitioned across multiple directory servers. This is accomplished by the use of Kerberos and URLs to allow crossing partitions located on different LDAP directory servers.
4. To insure the integrity of the information in the directory while allowing it to be easy to use. This has been attempted by:
   a. using Kerberos authentication along with the identity mapping APIs and impersonation of local user identities to control access to the data
   b. allowing directory entries to be stored and managed with enough flexibility to allow administrators to choose an LDAP directory server that meets their security needs
   c. providing directory management API's that perform directory searches and updates in a consistent manner.

Terms

Definitions of some LDAP schema terms used in this document.

"Schema" is borrowed from database practice, schema means meta-information, or information that describes other information. A Directory schema describes the kinds of objects (or entries) and their optional and required attributes (or properties) that can reside in a Directory. For example, a schema could define a person object class. The person schema might require that a person have a surname attribute that is a character string, specify that a person entry can optionally have a telephoneNumber attribute that is a string of numbers with spaces and hyphens, and so on.

"IBM Schema" defines the schema for use by IBM Customer Applications, IBM Applications and OS services that are LDAP enabled. The IBM Schema currently includes definitions for about 1400 object classes and attribute types defined by standards (e.g. X.501), RFC's (e.g. RFC 2256) and IBM development groups (e.g. On-Demand Server, OS/400, OS/390, AIX, etc.).

"DIT", Directory Information Tree, a hierarchical tree-like namespace comprised of directory entries managed by one or more directory servers. An example of a non-LDAP DIT is a Microsoft Windows file system's namespace. Typically a DIT is shown as a tree of connected nodes where each node is a directory entry. Each node is labeled using the nodes Distinguished Name (DN).

"DN", Distinguished Name, is a name (string) that uniquely identifies a directory entry in the DIT. A DN consists of pairs of attribute types and their values with each pair separated by a comma. A DN specifies the directory entry's name within an a hierarchical namespace (DIT) starting with the least significant part of the tree. Examples of two typical DN formats are; X.500 (cn=Joe Blow, ou=users, ou=Rochester, o=ibm, c=use) and Domain Component (cn=Joe Blow, ou=users, dc=rochester, dc=ibm, dc=com).

"Directory Entry", a object or entry in the directory that contains information about something. A directory entry is comprised of attributes (properties) where each attribute has a name, type and value. For example, a directory entry associated with a person may have an attribute called telephoneNumber, with a type of string and value of 555-1234.

"Object Class", part of the schema definition used by an LDAP server to limit the attributes that compromise a directory entry. The object class is a template that defines which attribute types are required and which are optional. When creating a directory entry, an application specifies a list of object classes to be used and thus defines the limits of what attribute types comprise the directory entry. There are three types of object classes; *abstract, structural* and *auxiliary.*

Abstract object classes define a set of attribute types and are used to derive (subclass) additional object classes. Derived classes 'inherit' attribute types from their parent. Abstract object classes are not used when creating directory entries. `Top` is an example of an abstract class that is used for sub classing other object classes.

Structural object classes define a set of attributes and are used when creating directory entries. `person` is an example of a structural class.

Auxiliary object classes define a set of attributes and are used when extending the set of attribute types of structural classes. `ePerson` is an example of a auxiliary class that can be used to extend the set of attribute types for the structural object class `person`.

"account" is user information that is related to a specific environment (operating system, application, etc.) and typically controlled by an administrator.

Overview

This section covers the DIT and schema associated with identity mapping. There are several characteristics of the DIT that are important to identity mapping and drive the need for several new schema items (e.g. object classes and attribute types).

- *"Multiple User Accounts"* Since a user may have mulitple accounts on one or more systems and each account is a separate directory entry, the DIT and schema must be able to manage multiple accounts for the same Enterprise User. Management of multiple accounts is done using **account pointers** that link together related directory entries. Account pointers are simply attributes containing URL values of account specific directory entries related to the user.

- *"Mapping Domains"* Since an enterprise wide directory may contain multiple domains of user and account information. Domains result from having multiple directory partitions from one or more directory servers, separate administrative domains, or simply separate containers of users. When locating account information in the DIT, the location of these domains is provided by a list of **domain pointers.** Domain pointers are simply attributes containing URL values for domains that contain accounts and user directory entries.
- *"Default DIT"* A default DIT is proposed to provide consistency across implementations of the identity mapping proposal and to fit with other directory usage proposals. Note that the administrator can elect to not use the default DIT and select their own DIT structure.
- *"Kerberos Authentication"* To provide consistent and transparent authentication across multiple LDAP directory servers, Kerberos based authentication is assumed. This requires a Kerberos SASL plug-in to be configured for both the LDAP client and the LDAP directory server. The Kerberos SASL combined with identity mapping will allow local security semantics to be imposed.

DIT Example (Single Server)

The following diagram shows an example of a DIT for identity mapping in a single server configuration. The purpose of this diagram is to introduce several of the identity mapping concepts using a simplified DIT. In the next section we provide a more complicated example using multiple servers to show a more complete identity mapping schema proposal.

The following can be noted about this example:

1. There is only one server (as4-1.rchland.ibm.com) and only one LDAP server running on that server. This LDAP server provides an LDAP repository and also 'projections' of local user accounts.
2. The DIT is comprised of two suffixes,
   - "dc=as4-1, dc=rochester, dc=ibm, dc=com"
   - "sys=OS/400, dc=as4-1, dc=rochester, dc=ibm, dc=com"
     one for each of the LDAP server backends that are required.
   a. The first suffix (shown on the left side of the DIT) is for the back end that implements an LDAP directory repository. Because of the security sensitivity of the directory entries under this suffix, if available, this back end would use local security to protect the directory entries.
   b. The second suffix (shown on the right side of the DIT) is for the back end that projects accounts from. These accounts are actually entries in a local user registry.
3. Under the first suffix is a default DIT structure comprised of:
   a. A default suffix of the computer's TCP/IP host name in domain component format.
   b. There are three default directory entries used as containers to hold additional directory entries associated with users, accounts and computers. Each of these containers provides an administrative domain (ACLs can be added to allow different administrators management access to the containers).
      i. In the DIT, under the *cn=Users* directory entry, Enterprise user information is found in directory entries defined with **`person`** and **`eUser`** object classes. These two object

classes are already part of the IBM Schema and are used today by several IBM products to define users.

ii. In the DIT, under the *cn=Accounts* directory entry, account information for the Enterprise User is found in directory entries defined with the `eAccount` and `KerberosAccount` object classes.
   1) In our example, Kerberos KDC repository are managed from the *cn=Accounts* directory by a Kerberos server, however this may not always be the case since 'projection' will be required if a Kerberos server uses a local registry.
   2) The `eAccount` directory entry is extended with the `MappedAccounts` auxiliary object class which provides links (`mappedAccountsURL`) to related accounts stored elsewhere in the DIT.
      a) There is a link to the joey (local OS/400 user profile) accounts that are being projected into the DIT (under a different suffix).
      b) There are account entries (defined by the `KerberosAccount` object class) containing Kerberos information for the Enterprise User.

iii. In the DIT under the *cn=Computers* directory entry, there is computer information associated with the specific computer. `eComputerSystem` and `eSystem` are both object classes that are already defined in the IBM Schema.
   1) The directory entry containing system information (`eSystem` in our example) is extended with the `MappingDomain` auxiliary object class.
      a) This directory entry is used to locate the various domains in the DIT that contain account mapping information (there is only one such domain in our example).
      b) This directory entry may also contain additional system information (not shown in our example).
      c) Note that directory entries defined with object classes other than `eSystem` can be extended by the `MappingDomain` auxiliary object class.

4. Under the second suffix in the DIT are the 'projected' directory entries. These directory entries are generated by a specialized LDAP server back end using account information managed by the local operating system.
   a. *Accounts* is a default directory entry used as a container to hold the account entries.
   b. Note that this suffix may have other directory entries used as containers to hold non-account information that is also being projected by the same LDAP server back end.
      i. In the DIT, under the *Account* directory entry are directory entries defined with the `OS400Account` object class. There is an account entry for each local operating system account (being projected).

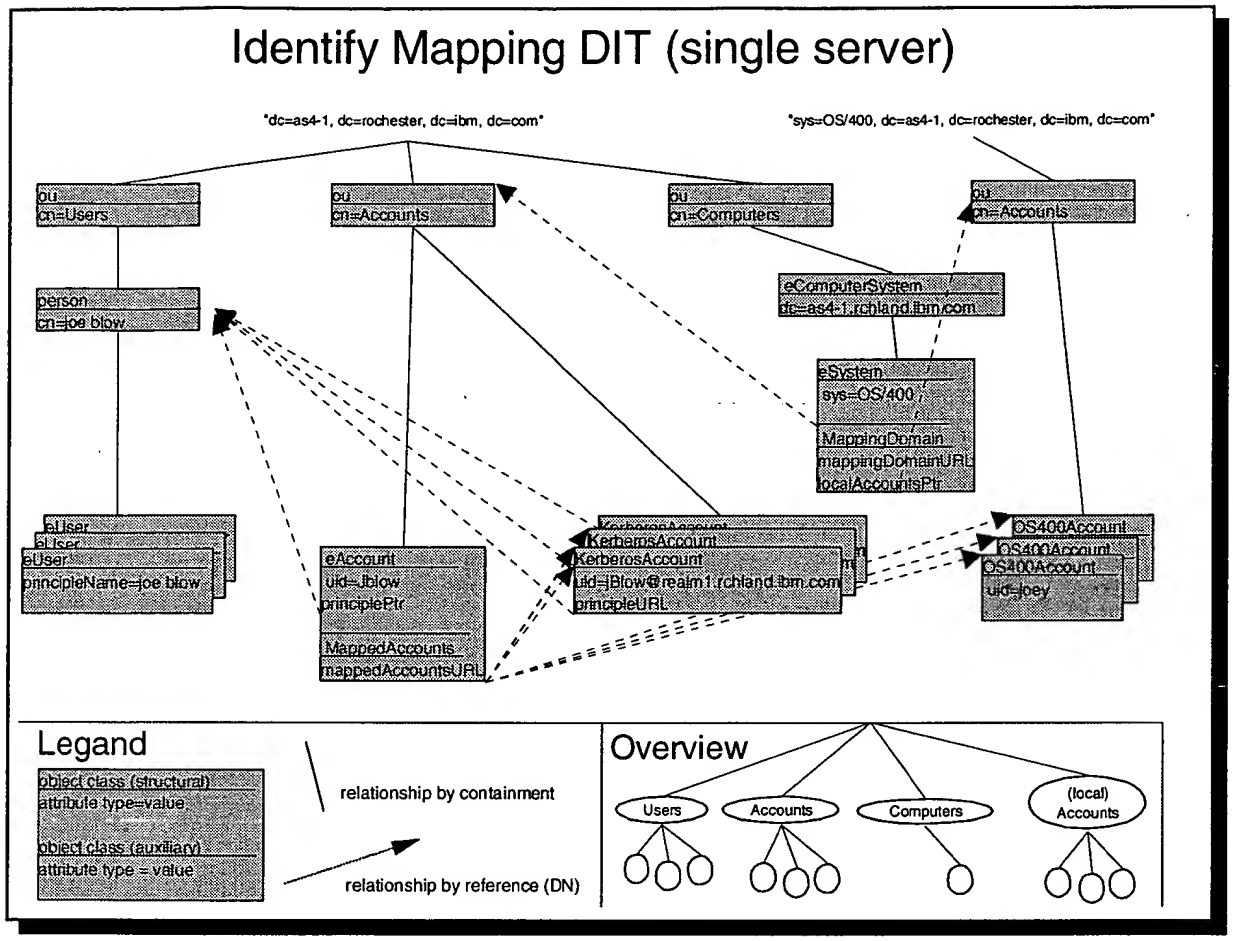

Figure 7. DIT for Identity Mapping (Single

Usage

In this section, we'll briefly describe how the DIT shown above is used for identity mapping.

Note that the functions outlined below would be implemented by new set of Identity Mapping Directory Management (IMDM) API's provided as part of this proposal and not by applications for the following reasons. These IMDM API's would be used by the Identity Mapping API's defined in this document and possible by applications. IMDM API's are required since:

1. *DIT updates and traversals are complicated by the number of directory entries and their relationships*
2. *Updates to the DIT must be done in a consistent manner to preserve the integrity of the relationships.*

Lookup the mapping domain information

This function, required by all IMDM API's, locates the mapping domain and optionally caches this information for later use. Below are the steps performed by this function:

1. bind to the LDAP directory server using a Kerberos service ticket for the LDAP directory server. The LDAP directory server host name and port can be obtained via a DNS lookup or a local configuration file.
2. obtain the URL's to the mapping domains by returning the values of the `MappingDomainURL` attribute of the appropriate directory entry (eSystem entry in our example). From our DIT example, the URL returned would be:
   `ldap://<hostname:port>/cn=Accounts, dc=as4-1, dc=rochester, dc=ibm, dc=com.`

Map a Kerberos identity to a local operating system user

This function is used to map from a Kerberos principle ID to another account of the Enterprise User, such as a local operating system account. Below are the steps performed by this function:

1. performs an LDAP bind to the LDAP directory server using a Kerberos service ticket for the LDAP directory server. The LDAP directory server host name and port can be obtained via a DNS lookup or a local configuration file.
2. performs an LDAP search of the directory container of each mapping domain for a list of accounts matching the Kerberos principle and realm passed into the API.
   a. The directory containers for the mapping domains is specified by the values of `MappingDomainURL` (see above).
   b. The search is performed against directory entries of type `MappedAccounts` where uid=principle@realm. From our DIT example above, this would be uid=jBlow@realm1.rchland.ibm.com.
3. checks the list of accounts returned for accounts corresponding to the local operating system.
   a. Locating the correct account(s) is done by comparing the value(s) of `mappedAccountsURL` for each of the account directory entries to see if they contain one of the values (DN's) of `localAccountsPtr`.
   b. In our example we would get a match on the directory entry "uid=joey, cn=Accounts, sys=OS/400, dc=as4-1, dc=rochester, dc=ibm, dc=com" because the directory entry "uid=Jblow, cn=Accounts, dc=as4-1, dc=rochester, dc=ibm, dc=com" has a MappedAccountURL that contains a DN value whose parent DN is "cn=Accounts, sys=OS/400, dc=as4-1, dc=rochester, dc=ibm, dc=com".

Other Functions

There are additional functions that IMDM API's will implement that still need to be defined.

DIT Example (Multiple Server)

The following diagram shows an example of a DIT for identity mapping in a configuration of multiple LDAP directory servers. Multiple LDAP directory servers are required when identity mapping is being used:

- with accounts projected for more than one environment.
- with accounts being stored in one LDAP directory repository on one computer system and accounts being projected for an environment from another computer system (shown below).

The diagram below shows a DIT where:

- On computer system aix.austin.ibm.com an LDAP directory repository contains:
  - Enterprise User white page information (directory entries of type person)
  - 'User' information (directory entries of type eUser)
  - Account mapping information (directory entries of type eAccount)
  - Kerberos accounts (directory entries of type KerberosAccount)
- Accounts are being 'projected' from the local operating system on as4-1.rochester.ibm.com.

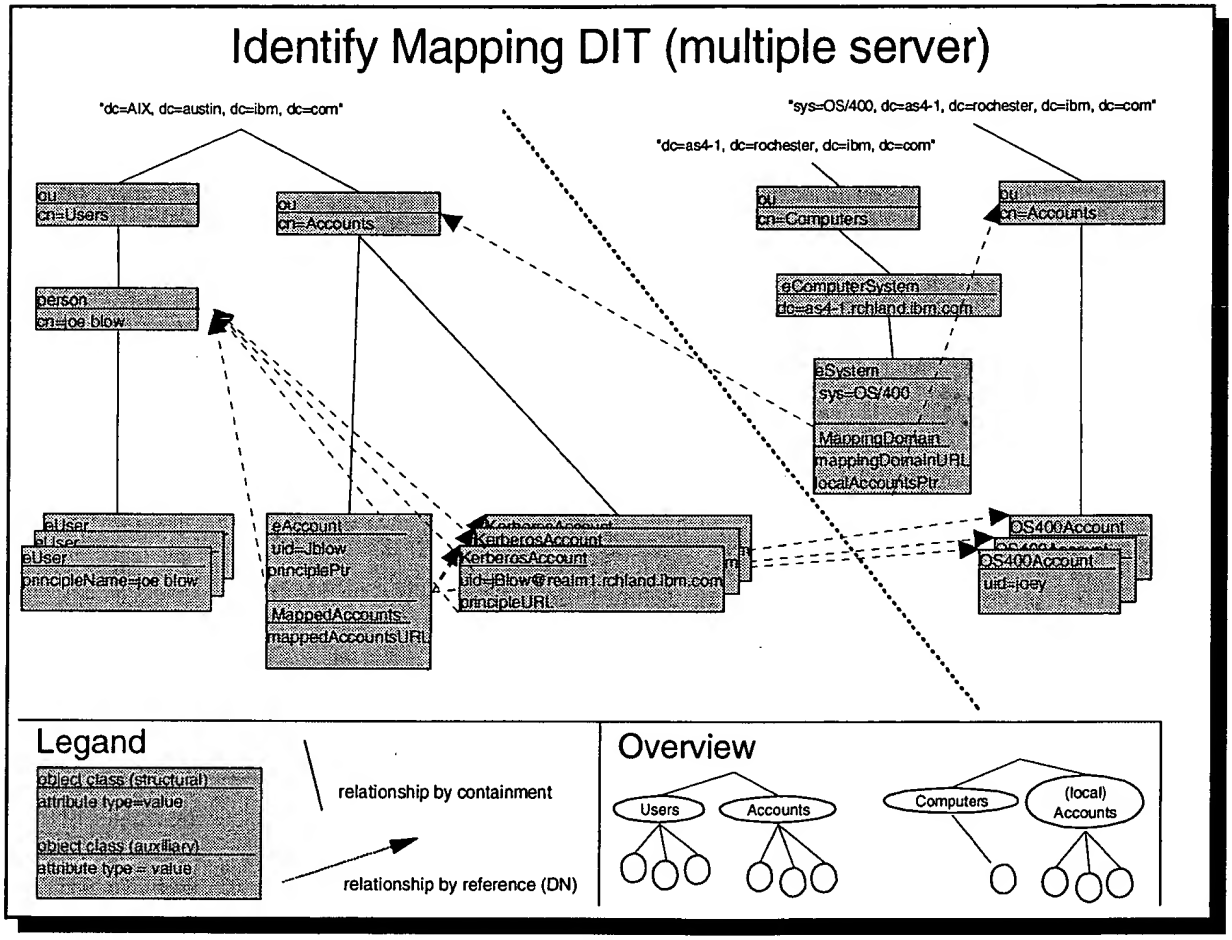

Figure 8. DIT for Identity Mapping (Multiple Server)

Usage

In a multiple directory server configuration (see above) the DIT is partitioned into multiple namespaces across more than one LDAP directory server. The key difference is that the attribute types that contain a URL provide the LDAP client application (IMDM API's) to locate a directory entry in a different LDAP directory. Note that Kerberos service ticket continues to be used to provided LDAP authentication across the different LDAP directory servers.

Object Classes and Attributes

Below are object classes and attributes associated with this proposal that need to be added to the existing IBM Schema. There are many object classes and attributes in the existing IBM Schema that would continue to be used as they are today. Note that the object class and attribute type names shown require a prefix of 'ibmCom1986-Common-' to be compatible with Microsoft schema conventions.

Schema Item Name	Type / Definition	Use	Set By	DIT location
MappingDomain	(new) Object Class	Provides links to mapped account entries elsewhere in the DIT. Specifically, this auxiliary class extends eSystem entries with a list of pointers (MappingDomainURL) to related containers of account mappings.	Applications or APIs that manage account relationships within a DIT.	Extends DIT entries that contain system related information. For example, an eSystem entry for a specific computer.
	(oid-tbd NAME 'MappingDomain' SUP top AUXILIARY MAY (MappingDomainURL))			
MappingDomainURL	(new) Attribute Type	A list of pointers (URL's) of related account mappings for a specific system.	See MappingDomain	N/A
	(oid-tbd NAME 'MappingDomainURL' SUP url)			
MappedAccount	(new) Object class	Provides a way to link account entries together. Specifically, this auxiliary class extends eAccount entries with a list of pointers (MappedAccountsURL) to related accounts located elsewhere in the DIT.	Applications or APIs that create accounts.	Extends DIT entries that contain account for a specific user.
	(oid-tbd NAME 'MappedAccount' SUP top AUXILIARY MAY (MappedAccountsURL))			
MappedAccountsURL	(new) AttributeType	A list of pointers (URL's) of related account entries for a specific account.	See Mapped Account.	N/A
	(oid-tbd NAME 'MappedAccountsURL' SUP url)			
LocalAccounts	(new) Object Class	Provides links to directory containers of accounts for the local system. Specifically, this auxiliary class extends eSystem entries with a list of pointers (LocalAccountsPtr) to containers of local accounts.	Could be initialized by the LDAP server that provides the container (s) of local accounts or can be managed by an application or API.	Extends DIT entries that contain system related information. For example, an eSystem entry for a specific computer.
	(oid-tbd NAME 'LocalAccounts' SUP top AUXILIARY MAY (LocalAccountsPtr))			
LocalAccountsPtr	(new) Attribute Type	A list of pointers (DN's) to containers of local accounts.	See LocalAccounts	N/A
	(oid-tbd NAME 'LocalAccountsPtr' SUP distinguishedName)			

Appendix B

Low-Level Enterprise Group Schema

This section is copied verbatim from Lee Rafalow's Enterprise User workgroup paper.

The creation of enterprise groups and roles provides a common representation of the intended membership relationships on each of the participating systems. Any user (person, service, service access point, group, role, etc.) that is both a member of an enterprise group and that has an identity (account) on a participating system needs to be added to the related group on that participating system.

An additional class is defined to provide participating systems information needed to take the intersection of enterprise group membership and system account.

(oid-tbd NAME 'ibm-MappedGroup' SUP top AUXILIARY
MAY (ibm-ParticipantPtr))

(oid-tbd NAME 'ibm-ParticipantPtr' SUB distinguishedName)

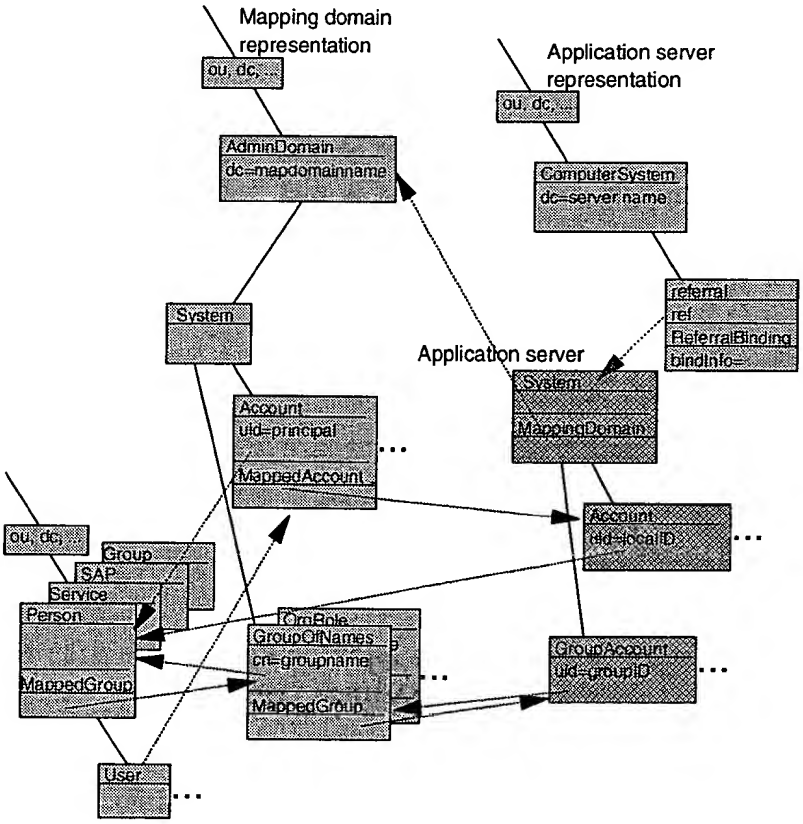

Figure 9: Enterprise Groups

In figure 9 above, looking at the groupOfNames instance, the ibm-MappedGroup auxiliary extension provides the ibm-ParticipantPtr values that associate the enterprise group (or role) with

system groups. A similar reverse pointer may be maintained by the systems to track their mapping to enterprise groups.

In adding a member to an enterprise group, the management tool goes through the following steps.

1. The person (service, etc.) entry is extended, if not already present, with the ibm-MappedGroup auxiliary class and both the group member (or role roleOccupant) attribute value and the ibm-ParticipantPtr attribute value are updated to point to one another. This establishes membership in the enterprise group and, for those products that exploit the common directory representation of user, account and membership this is sufficient.
2. For those products that maintain their own groups and roles, the management tool must then find, using the ibm-User accountHint dn, the enterprise mapped account for this user and compare the ibm-MappedAccountsPtr values and the ibm-ParticipantPtr values. The DNs are parsed into their parent ibm-System class DNs which are then compared. Matches indicate that the user has an account on a participating system.
3. Using LDAP and the binding information provided in the ibm-ReferralBinding, the management tool must then add the user's local system identity (e.g., ibm-Account entry) to the group on the local system. In the event of a failure during this operation, recovery may be initiated by the management tool or by the system upon reestablishment of contact (no transaction support is provided in LDAP).

When defining groups (or roles) on a local system, current system interfaces may be used. In addition, the creation of the appropriate group or role object entry on the application server via the LDAP protocol should also affect the same group creation operation.

Appendix C

Enterprise User Identity Mapping and Dascom

Both the Enterprise User Management product and this Server Group Cross-Platform Identity Mapping Proposal focus on the problems associated with the Identification and Authentication (I&A) of users within an enterprise. But user I&A, by itself, is only part of an effective security model. Access control, which is usually based on Access Lists (ACLs) that are associated with specific sets of resources, is required for a complete security model. Like the disparity in I&A implementations, access control implementations also differ across IBM platforms and middleware. WebSphere requires a "Unified Security Model" for distributed applications that are based on Component Broker and Java. This requirement has led IBM to acquire Dascom with the intention to implement distributed access control services across IBM distributed applications and systems. The objective is that these services, which are embodied by the Dascom authorization APIs, will be the common foundation for distributed access control function .

Dascom technology is said to be "authentication agnostic" in that, theoretically, authorization and access control can be separated, and therefore Dascom access control technology doesn't care which authentication technology is implemented along with it. In practice, however, Dascom access control technology requires the support of certain specific identity related functions and constructs, such as the notion of Universal Userids (UUIDs) that represent individual users and specific groups of users in Dascom access control lists. So, although Dascom is authentication agnostic, it none-the-less drives certain behavioural requirements on authentication technology, and this fact will eventually need to be considered in parallel with the other objectives and messages of these papers regarding identity management and cross-platform user and group identity mapping.

Appendix D

More on Operating System Integrity

It is extremely difficult, if not impossible, for an application (i.e. any program that is not part of the operating system) to define and enforce the security semantics for operating system resources without creating operating system integrity exposures. This is because any new semantics introduced by the application are ultimately built on top of OS provided storage and semantics.

Secureway Directory, for example, has defined security semantics for the directory. If the operating system creates a new resource managed in the directory as it is implemented today, we create a system integrity problem. Why? There are two reasons. First, the directory has it's own set of security semantics which are different from the operating system's. The semantics provided by the directory are not as rich as most operating systems and therefor the operating system cannot enforce the semantics it would otherwise enforce. Second, the directory is built on top of operating system (at least in the case of OS/390 and AS/400) provided function -- DB2 -- that have a different set of interfaces and semantics for manipulating resources. So there are two sets of completely different interfaces, each providing it's own set of semantics, and neither of which is what the operating system may have otherwise enforced.

These sorts of design issues is what leads us to the secure LDAP back end rather than using the standard LDAP back end repository.